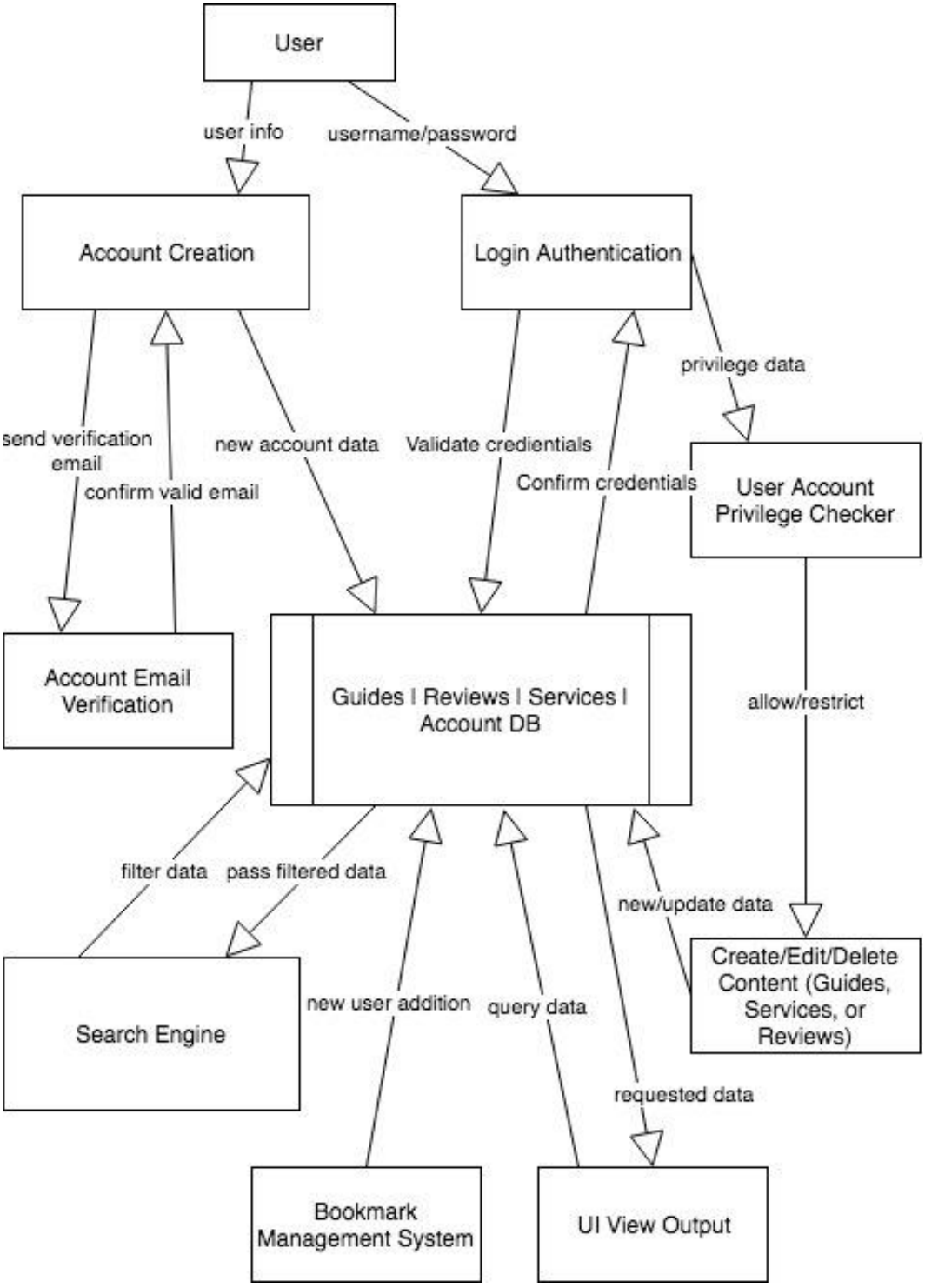
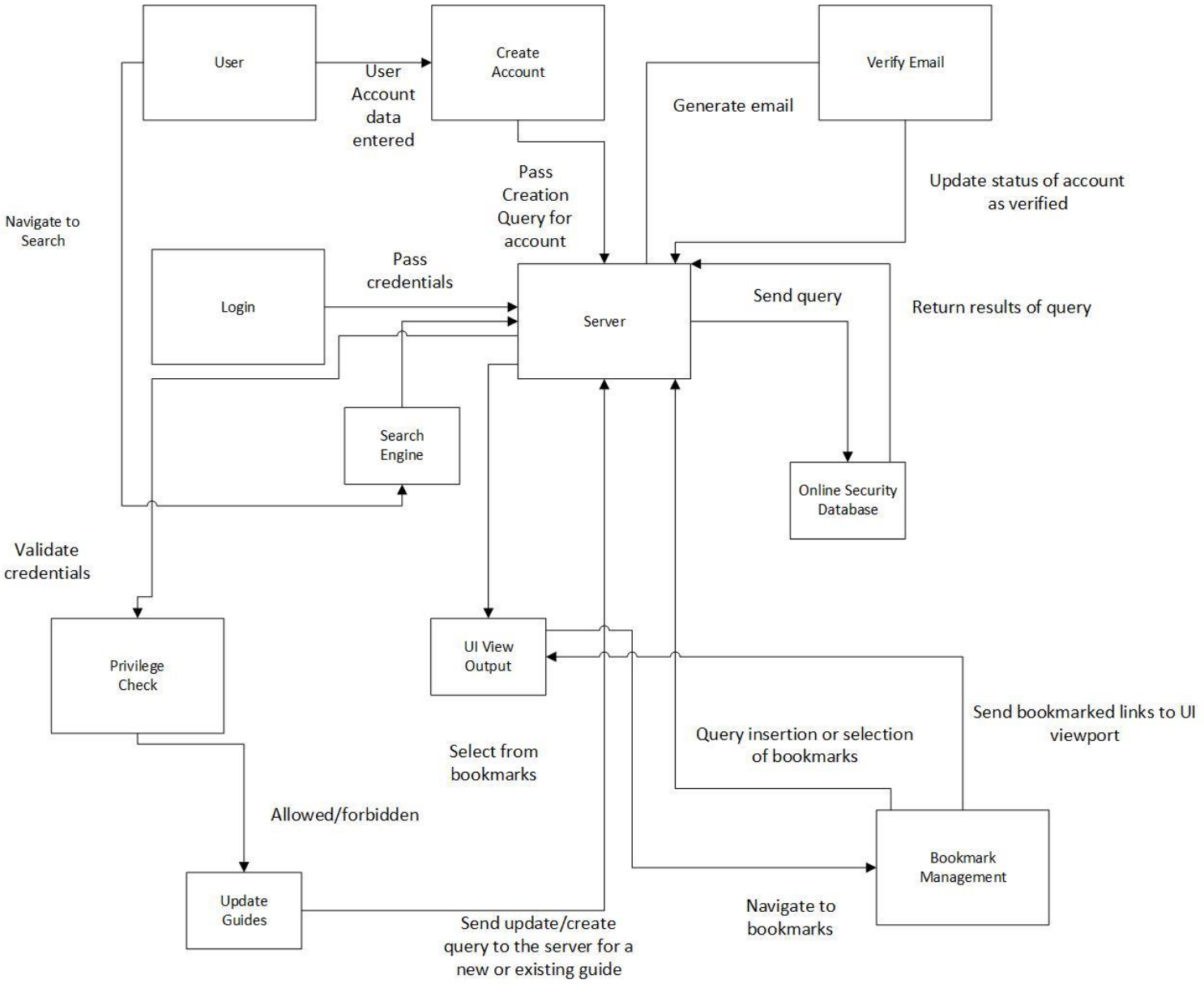
CS361 - Group 6 - HW3: Architecture Assignment

Sheng Bian | Michael Waldrop | Connor Shields | Linge Ge

David's High-level Architecture of System



Connor's High-level Architecture of System



Identification of key quality attributes:

Analysis of quality attributes:

Maintainability:

David's:

Some of the low-level inner workings of David's architecture are unclear, which could make it difficult to maintain both for while debugging and updating. Specifically, the architecture doesn't tell us how the data is being stored. Instead, it seems to show that each page acts as its own servlet, but an engineer might also assume that data should be passed to the database through the standard client-server interaction. However, the interaction between client-pages and the server are very well-defined, meaning that it would be reasonably straightforward for a team of engineers to implement the front-end with the same approach in mind.

Connor's:

Connor's architecture specifies how the data will be stored, as well as how it will be passed to and from the server and the client pages. This makes the entire structure of the operation clearer, which means that it will be easier to maintain over the lifetime of the project.

Efficiency:

Both architectures implement proven models of efficiency for hosting websites: the database stores the data, the server queries this data with requests from web pages, and then the client side is rendered from the information the server finds in the database. The structures are laid out to the standards of best practices for database-server-client interactions, so they will set up any engineering team very well to implement the project to be as efficient as possible.

Reliability:

As mentioned in the failure cases, the main concerns regarding reliability are the issues of login credential failure, consistency of updates written to the database, and a UI that supports filtering like the user would expect. Because the data is stored permanently in a database, it's unlikely that the user will see their data passed incorrectly. However, David's architecture would potentially pose a greater risk for this failure case, because the independent servlets could act simultaneously without any validation of data from the server.

Integrity:

As we mentioned regarding efficiency, both architectures give engineers the model they need to properly lay out a more secure website. Storing the data from a website in a protected database is a proven method of securing information, so from here, the standard of security that our

project achieves depends entirely on how well the structure is actually implemented during the programming phase and architecture decomposition.

Usability:

Both architectures optimize usability by laying out an intuitive interface. David's and Connor's design show a clear progression of data flow for both users who will use the site without an account in order to search data and those who will update and rate guides.

Testability:

David's and Connor's architectures can be tested by comparing custom search and update results from the client-side against manually-written queries, which would ensure that the information is being passed back and forth correctly. Client-to-client interaction can be tested simply by checking links and verifying that the UI matches the mockup.

Flexibility:

Both architectures are designed for optimal flexibility, as they allow users to update and create information, which then becomes available for other users to interact with.

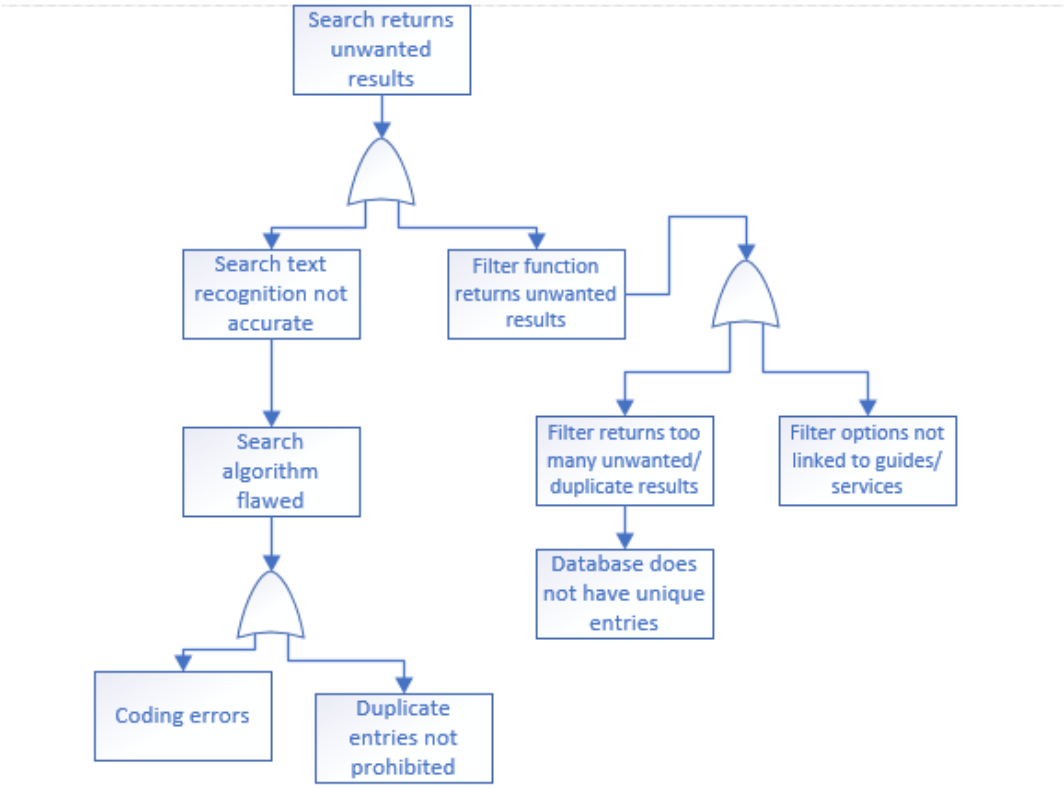
Portability:

David's architecture might not be as portable, because the interaction of servlets with a database could require a more unique layout. Connor's architecture is as portable as any website involving a server. The code should be able to transfer to any other server-database combination, as long as the machine meets the system requirements and supports the languages and libraries required by the project. If needed, the same setup can be rewritten to use available libraries and languages (e.g., if it were originally written in Node.js, there's no reason it couldn't be recreated in the .NET framework or with PHP).

Interoperability:

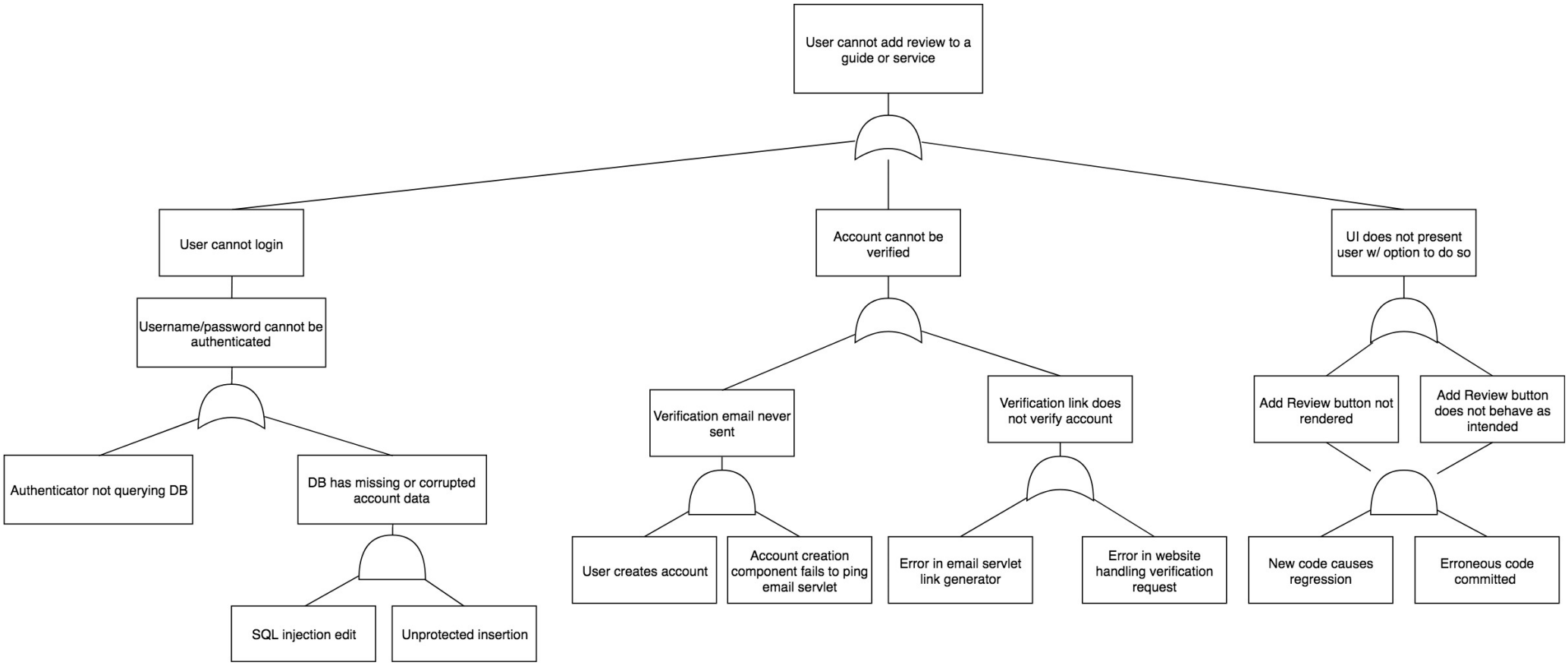
Both architectures again are setup to give the design team everything they need to make the service interoperable. The main concern for this service would be interoperability with different web browsers, but this is something that can be detailed further in architecture decomposition.

Failure Mode #1:



Of the two architectures, the key distinction is that Connor's uses the server to connect all components, including the database, while David's uses the database as the central connection point for the architecture. By routing requests through the server, it is easier to identify specific faults related to one component or another. Directly connecting to the database can pose a number of difficulties in regard to possible coding errors as well as security issues, while routing through the server allows for at least one extra layer of protection from such problems. Needing to go through the server for any query also allows for different parts of the website to be compartmentalized for the purposes of debugging faulty code. If you encounter an error when accessing the database in David's architecture, for example, then there could be an issue with the database, or it could be a completely different issue related to your connection. By going through the server first, you can confirm that you are able to access the server successfully before you make a database query, which helps to narrow down problems in that event that the website is not functioning correctly.

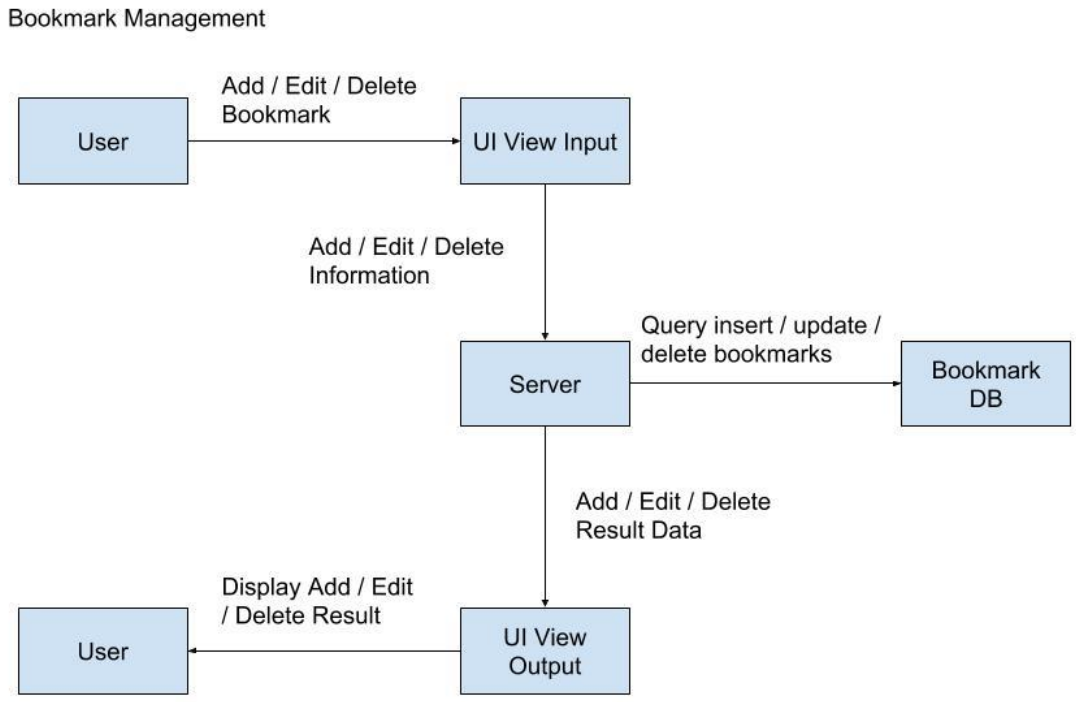
Failure Mode #2:

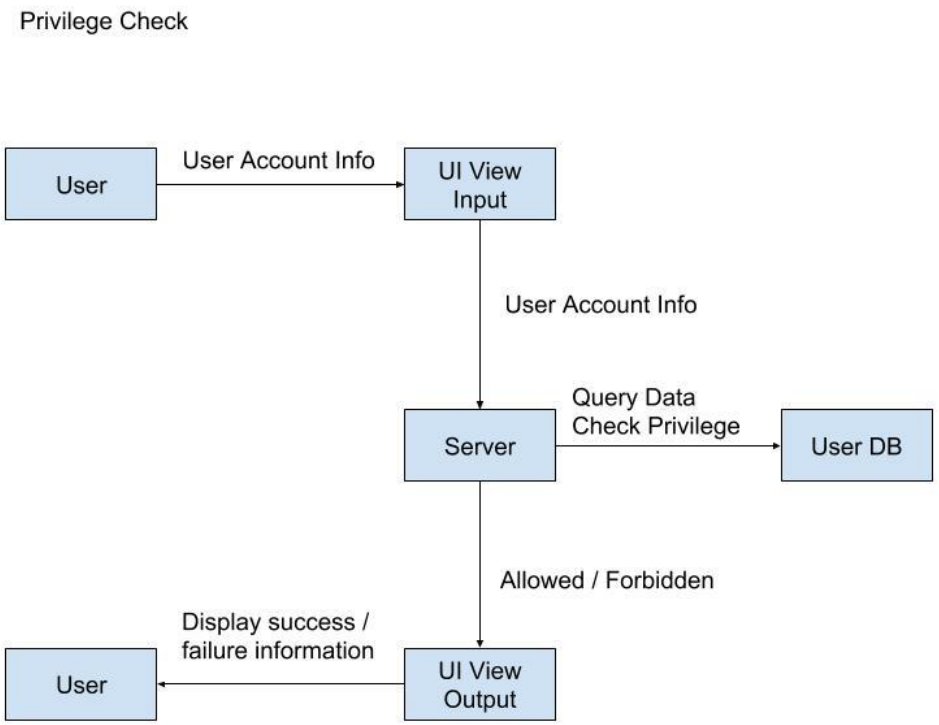


One of the main features of the website is that users are able to add reviews to the guides and services which are offered. The inability to do so represents a huge failure mode of the site. The three primary ways (and each has their own sub-failure reasons) for the inability to write and rate reviews are: user cannot login, account cannot be verified, and UI doesn't allow the user to do so. All of these issues can be traced back to problems with components that handle each logic, and these components reside on the server. David's architecture is more prone to have this failure mode, because the architecture does not distinguish a distinct server. Each component which can lead to the failure (ex: login, verification, UI output) is mentioned, but they exist as independent servlets without a centralized system such as a main server providing overhead communication. Each component servlet simply passes data to the database and act nearly independently. This design would be more prone to fail due to lack of overhead.

lower-level dataflow diagram

We select Connor's high-level architecture for further decomposition. The two lower-level dataflow diagrams are for bookmark management and privilege check.





Validate Selected Architecture

Use Case #1 walkthrough (Connor's architecture):
Name: User creates an account after navigating to the website for the first time.
Actor: User
1. User navigates to the website home page.
   - Not technically represented in the diagram. It is assumed the user will start out on the home page by default.
2. User selects a Create Account redirect at the top of the web page.
   - Represented by the Create Account event.
3. A webform with input fields opens. The text description for the required information in appears on a line above each form. the fields are:
○ Text: Full Name
Field: a text input field
○ Text: Email
Field: a text input field
○ Text: Phone Number
Field: a text input field

○ Text: Technical Savvy
Field: a dropdown with the options: “Technical Expert”, “Above Average” “Some Technical Savvy”, and “What’s ‘Technical’ mean?”. The dropdown is set on a blank input until the user clicks on the arrow to select the option.
○ Text: Username
Field: a text input field
○ Text: Password
Field: a text input field
○ Text: Confirm Password
Field: a text input field
- Represented by the data insertion connector attached to the Create Account event.

4. The user fills out each field. The following conditions apply when filling out the Fields:
○ The Username must be unique. Otherwise, they will not be able to proceed.
○ The email must be a valid email address. It must have text, followed by @, followed by text, followed by a period, which is then followed by com, net, edu, org, or gov.
○ The password must match in both password input fields (case sensitive). Otherwise, the user will not be able to proceed.
○ No input fields can be blank. Otherwise, the user will not be able to proceed.
- Also represented by the data insertion connector.

5. The user clicks a Submit button at the bottom of the form.
- Also represented by the data insert connector.

6. The user goes to the email inbox of the email address that they used on the form.
- Represented by the connector/event for email verification.

7. The user will see an email from our website.
- Represented by the verify email event.

8. The user opens the email.
- Also represented by the verify email event.

9. The user sees text that reads: “Thank you for creating an account with Privacy for the Average Citizen. Please verify your account by following the link below”, as well as a button, below the text, that says “verify my account.”
- Same as the previous step.

10. The user clicks on the button.
- Represented by the transition from the Verify Email event to the update status connector.

11. The user is redirected to a webpage that says: “Your account has been verified! You can now access our content with your username and password at the login page.”
- Same as above.

<u>Use Case #2 walkthrough (Connor’s architecture):</u>
Name: User wants to learn about a type of service
Actor: User
1. User navigates to the website home page

- Not represented physically. It is expected the user will start on the home page by default.

2. User sees the box of links to different categories of guides, such as VPNs, Ad/Script Blockers, etc.
   - Home page links would be present on the home page, which is not physically represented in the architecture.
3. User picks the desired guide category by clicking on the link and waiting for the page to Load
   - Represented by the Select data fields connector.
4. This redirects to a new page with entries in that category, for example a page with entries about VPNs
   - This could be represented by the Search Engine event.
5. User decides how to sort the entries, which can be by most popular or alphabetically
   - This is represented by the Search Engine/filter query connector.
6. User clicks on the sorting method
   - Same as the previous step.
7. User looks through the entries
   - Represented by the server response into UI View Output event.
8. User chooses which entry he/she wants and clicks on whichever entry they chose
   - This is represented by the database query. While not directly connected to the UI View Output event, it is tied to the server.
9. This redirects to a new page with the guide that was chosen
   - Tied to database query.
10. User reads the guide and is informed on the category he/she chose
    - Same as the previous step.

<u>Use Case #3 walkthrough (Connor's architecture):</u>

Name: User rates a service and leaves a user review on the Online Privacy website
Actor: User

1. User navigates to the website home page.
   - Not physically represented. Expected by default.
2. User selects a Login redirect at the top right of the web page.
   - Represented by the Login event box.
3. A text entry box opens.
   - Same as the previous step.
4. User is prompted to enter unique username and password.
   - Same as the previous step
5. User enters username and password and clicks the Submit button to log in.
   - Represented by the connector arrow between Login and Server
6. User is presented with a list of bookmarked services, sorted by type, saved after previous interactions with the website search function.
   - Represented by Bookmark management event.
7. User clicks the "Rate and Review" redirect button next to a previously bookmarked VPN Service.

- Represented by the privilege check event, as only users can rate and review.

8. User is taken to a form which allows them to click a 1 – 5 star rating for the product, and a text entry box that allows them to type their review.
    - Same as previous step.
9. User clicks on the fourth star from the left, which “fills” four stars and gives the service a 4/5 rating.
    - Same as previous step.
10. User types out their review of the service in the provided text box and clicks the Submit button to finish.
    - Could be represented by the Create/Update guides event, as it works the same for guides and services.
11. The user review is posted on the specific page for the service, marked with their account name as the reviewer.
    - Same as above.

Implications and revisions

There are a few modifications that could be made to the selected architecture:

1. User needs to interact with the website through the homepage. Currently, the architecture doesn’t model the need for the user to launch other subpages on the site through the main homepage. Instead, it is just left to assumption, but this could be a source of confusion for the team implementing this project. The architecture should be revised so that everyone is on-board with where the user will land when navigating to our site, and how the user will access the rest of the content through that homepage. This will improve maintainability and potentially usability.
2. Currently, the architecture doesn’t show exactly how the user will filter the data when they are searching for information to view and/or update. While this can be explained further in architectural decomposition, it is something that could be delineated in more detail in the dataflow of the architecture model. This would improve maintainability and testibility.
3. The architecture model doesn’t specify how step 10 from the 3rd use case should be implemented, so that is one more aspect that needs a more detailed modelling. Elaborating on how exactly the user can expect to navigate to a guide, create a service/guide, and how the system will save the user’s changes to that guide or service will ensure more consistency when we delegate the individual tasks of programming this software service.

There are a few changes that should be made to the architecture so that it conforms more accurately to the use cases, but on the whole, it models the customer’s vision and gives a solid foundation for our team to organize and continue developing the project.

Briefly summarize the contribution of each of your team members

- David was working on one data flow diagram of high-level architecture and one failure mode.
- Connor was working on one data flow diagram of high-level architecture, identify key attributes of system and explain the implications.
- Michael was working on one failure mode and walking through use cases.
- Sheng was working on 2 lower level data flow diagram and final merge of document.